Verification Conditions for Procedure Calls (Mathematical Studies of Information Processing)

AUTHOR(S):

ARAKI, KEIJIRO; USHIJIMA, KAZUO

CITATION:

ARAKI, KEIJIRO ...[et al]. Verification Conditions for Procedure Calls (Mathematical Studies of Information Processing). 数理解析研究所講究録 1982, 454: 138-159

ISSUE DATE:

1982-04

URL:

http://hdl.handle.net/2433/103005

数理解析研究所講究録
454巻 1982年 138-159

138

Verification Conditions for Procedure Calls

Keijiro ARAKI and Kazuo USHIJIMA

Department of Computer Science
and Communication Engineering
Kyushu University
Hakozaki, Higashi-ku
Fukuoka 812, JAPAN

Abstract

In this paper we discuss axiomatic proof of procedure calls. We deal with a Pascal-like programming language with
(a) arrays, pointers and records;
(b) assignment statements, compound statements, if statements, while statements and procedure calls including predeclared procedure new.

Procedures may contain recursive calls and have both variable parameters and value parameters. We place two restrictions on procedures.
(1)Global variables are not allowed to appear in any procedures.
(2)All parameters of procedures other than new must be of integer type.
These restrictions lead concise and clear proof rules, through which we obtain concise verification conditions for procedure calls. They can be applied easily, and are readily implemented in verification systems.

1. Introduction

We have presented a formal logical system [Araki-79] for proof of partial correctness of Pascal-like programs. Those programs consist of assignment statements, compound statements, if statements, while statements, and calls of the predeclared procedure new. Programs deal with data of integer type, array types, pointer types, and record types. According to the logical system, we have implemented a verification condition generator [Araki-80]. It automatically generates verification conditions for partial correctness of the Pascal-like programs.

However we cannot deal with procedures other than new in the logical system mentioned above. In this paper we present proof rules for procedure calls so as to prove partial correctness of programs with more general procedure calls.

Procedures may include recursive calls, and have both variable parameters and value parameters. We place two restrictions on procedures.

1) Non-local variables are not allowed to appear in any procedures.

2) All parameters of procedures other than new must be of integer type.

These restrictions lead clear and concise proof rules, which can be applied easily to program verification. We can obtain verification conditions for procedure calls through these proof rules. If we prove the verification conditions true, then the partial correctness of programs are established. Verification condition generators for programs with procedure calls are readily implemented owing to the clarity of the proof rules.

In section 2, we describe briefly the notations used in this

paper. In section 3 we discuss procedure calls, and present proof rules for them. In section 4 we obtain verification conditions for procedure calls. In section 5 we present applications of the proof rules to proving partial correctness of programs with procedure calls. Concluding remarks are given in section 6.

2. Notations

Most of the notations used in this paper are defined in [Araki-79]. The important of them are summarized in this section. The newly introduced notations concerning procedures are also described below.

Partial correctness assertion

$$P \{ S \} Q$$

Here P and Q are logical formulas, and S is a program. This notation means "if P is true before initiation of the program S, then Q will be true on its completion." P is called the input condition for S, and Q is called the output condition for S.

Weakest antecedent

$$[S] Q$$

Here Q is a logical formula, and S is a program. It means "after executing the program S, Q holds." Any formula logically equivalent to [S]Q is called a weakest antecedent of Q via S. If $P \supset [S]Q$ is proved, then the partial correctness assertion $P\{S\}Q$ is proved. Therefore any formula which logically implies $P \supset [S]Q$ is called a verification condition for partial correctness of the program S with respect to the input condition P and the output

condition Q. We get a verification condition by transforming the output condition via the program. The transformation rules have been presented in [Araki-79].

Procedure declaration

```
procedure pr(var x1,...,xk:integer;
                y1,...,ym:integer);
          B
```

Here pr is a procedure identifier, x1,...,xk are formal variable parameters, y1,...,ym are formal value parameters, and B is a procedure body. All parameters are of integer type. Non-local variables cannot appear in any procedures.

Procedure call

```
pr(v1,...,vk,t1,...,tm)
```

Here v1,...,vk are actual variable parameters, and t1,...,tm are actual value parameters. v1,...,vk should be distinct variables. t1,...,tm are expressions of integer type which may include actual variable parameters v1,...,vk.

3. Procedure Call Proof Rules

According to [Hoare-73], a procedure call pr(v1,...,vk,t1,...,tm) is considered to be equivalent to the sequence of assignment statements (executed simultaneously):

$$\begin{cases} v1 := f1(v1,\ldots,vk,t1,\ldots,tm) \\ \quad \cdots \\ vk := fk(v1,\ldots,vk,t1,\ldots,tm) \end{cases} \qquad (3\text{-}1)$$

f1,...,fk are regarded as the functions which map the initial values of the actual parameters v1,...,vk,t1,...,tm on entry to the procedure onto the final values of v1,...,vk on completion of the execution of the procedure body B.

Instead of writing down these functions f1,...,fk, however, we describe the meaning of execution of the procedure body B by a partial correctness assertion as follows.

$$\bigwedge_{j=1}^{m} yj=dj \wedge IN(x1,...,xk,y1,...,ym)$$

$$\{ B \} \qquad (3\text{-}2)$$

$$OUT(x1,...,xk,d1,...,dm)$$

Formulas IN(x1,...,xk,t1,...,tm) and OUT(x1,...,xk,d1,...,dm) represent the meanings of the functions f1,...,fk. The condition IN represents a relation between the initial values of the parameters on entry to the procedure. The other condition OUT represents a relation between the result values of the variable parameters and the initial values of the value parameters, which are denoted by fresh variables d1,...,dm.

Now, we consider a procedure call

pr(x1,...,xk,y1,...,ym)

in which the names of the formal parameters have been fed back as actual parameters. It is fairly obvious that this call has the same effect as the execution of the procedure body itself. Thus we obtain the following rule.

$$\frac{\begin{array}{l}\underline{\text{procedure}}\ \text{pr}(\underline{\text{var}}\ x1,\ldots,xk : \text{integer};\\ \qquad\qquad y1,\ldots,ym : \text{integer});\\ \qquad B\ ,\\ \bigwedge_{j=1}^{m} yj=dj \wedge IN(x1,\ldots,xk,y1,\ldots,ym)\\ \qquad \{\ B\ \}\\ \qquad OUT(x1,\ldots,xk,d1,\ldots,dm)\end{array}}{\begin{array}{l}\bigwedge_{j=1}^{m} yj=dj \wedge IN(x1,\ldots,xk,y1,\ldots,ym)\\ \qquad \{\ pr(x1,\ldots,xk,y1,\ldots,ym)\ \}\\ \qquad OUT(x1,\ldots,xk,d1,\ldots,dm)\end{array}} \qquad (3\text{-}3)$$

Consider next the more general call

$$pr(v1,\ldots,vk,t1,\ldots,tm)\ .$$

This call is intended to perform upon the actual parameters v1,...,vk,t1,...,tm exactly the same operations as the body B would perform upon the formal parameters x1,...,xk,y1,...,ym. Thus, if the condition IN(v1,...,vk,t1,...,tm) holds before the procedure call, the condition OUT(v1,...,vk,d1,...,dm) is expected to hold after execution of the call. Here variables d1,...,dm denotes respectively the values of t1,...,tm immediately before the procedure call. This reasoning leads the rule

$$\begin{array}{l}\bigwedge_{j=1}^{m} yj=dj \wedge IN(x1,\ldots,xk,y1,\ldots,ym)\\ \qquad \{\ pr(x1,\ldots,xk,y1,\ldots,ym)\ \}\\ \qquad OUT(x1,\ldots,xk,d1,\ldots,dm)\end{array}$$

$$\frac{\bigwedge_{j=1}^{m} tj=dj \wedge IN(v1,\ldots,vk,t1,\ldots,tm)}{\{\ pr(v1,\ldots,vk,t1,\ldots,tm)\ \}\quad OUT(v1,\ldots,vk,d1,\ldots,dm)\ .} \qquad (3\text{-}4)$$

Above two rules correspond to the rule of invocation and the proposed rule of substitution in [Hoare-71]. In this paper, actual value parameters can contain actual variable parameters, while in [Hoare-71] they cannot.

The rules given above are, however, not sufficient for the proof of the properties of recursive procedures. The reason is as follows. The body of a recursive procedure contains at least one call of itself. When we want to prove the second premise in (3-3) so as to establish the partial correctness of a procedure call (the conclusion of (3-3)), we encounter a call of itself, and must prove the partial correctness of the recursive call, which we must prove by means of the rule (3-3). Thus we must prove again the partial correctness of the body as the second premise. And the proof will continue forever.

The solution to the infinite regress is simple: to permit the use of the desired conclusion as an induction hypothesis in the proof of the body itself. Therefore, the rule (3-3) should be replaced by the following.

```
procedure pr(var x1,...,xk : integer;
                 y1,...,ym : integer);
        B ,
```

$$\frac{\begin{array}{l}\bigwedge_{j=1}^{m} yj=dj \wedge IN(x1,\ldots,xk,y1,\ldots,ym) \\ \quad \{\ pr(x1,\ldots,xk,y1,\ldots,ym)\ \} \\ \quad OUT(x1,\ldots,xk,d1,\ldots,dm) \\ \vdash \bigwedge_{j=1}^{m} yj=dj \wedge IN(x1,\ldots,xk,y1,\ldots,ym) \\ \quad \{\ B\ \} \\ \quad OUT(x1,\ldots,xk,d1,\ldots,dm)\end{array}}{\begin{array}{l}\bigwedge_{j=1}^{m} yj=dj \wedge IN(x1,\ldots,xk,y1,\ldots,ym) \\ \quad \{\ pr(x1,\ldots,xk,y1,\ldots,ym)\ \} \\ \quad OUT(x1,\ldots,xk,d1,\ldots,dm)\end{array}} \qquad (3\text{-}5)$$

When we want to prove partial correctness of mutual recursive procedures, we add the desired conclusions of all procedures that are called in the procedure bodies to the second premise as induction hypotheses.

4. Verification Conditions for Procedure Calls

Using the proof rules presented in the previous section, we prove the partial correctness of a procedure call

$$P\ \{pr(v1,\ldots,vk,t1,\ldots,tm)\}\ Q\ .$$

Besides those proof rules, we use the rules of consequence

$$\frac{P \supset R\ ,\ R\{S\}Q}{P\ \{\ S\ \}\ Q} \qquad \text{and} \qquad \frac{P\{S\}R\ ,\ R \supset Q}{P\ \{\ S\ \}\ Q}$$

and obtain verification conditions for procedure calls. We establish the partial correctness by proving the verification condi-

tions.

The verification condition is written as

$$P \supset (IN(v1,\ldots,vk,t1,\ldots,tm) \wedge (OUT(w1,\ldots,wk,t1,\ldots,tm) \supset [v1:=w1;\ \ldots\ ;\ vk:=wk]\ Q)) \quad (4\text{-}1)$$

By separating the above, we consider the following two formulas.

$$P \supset IN(v1,\ldots,vk,t1,\ldots,tm) \quad (4\text{-}2)$$

$$P \wedge OUT(w1,\ldots,wk,t1,\ldots,tm) \supset [v1:=w1;\ \ldots\ ;\ vk:=wk]\ Q \quad (4\text{-}3)$$

The formula (4-2) requires that the condition IN in the input condition for the procedure body should hold before the procedure call. That IN(v1,...vk,t1,...,tm) holds assures that the output condition OUT for the procedure body will hold after execution of the call.

The formula (4-3) requires that the condition OUT(w1,...wk,t1,...,tm) should imply the formula Q, where OUT is the relation between the new values of the variable parameters after execution of the call and the initial values of the value parameters. Here the initial values of the value parameters are denoted by t1,...,tm themselves, while the new values of the variable parameters v1,...,vk are denoted by fresh variables w1,...,wk. Assignment statements

$$v1:=w1;\ \ldots\ ;\ vk:=wk$$

represent that the values of the actual variable parameters are changed after execution of the procedure call.

We must also prove the partial correctness assertion of the

procedure body

$$\bigwedge_{j=1}^{m} yj=dj \wedge IN(x1,\ldots,xk,y1,\ldots,ym)$$

$$\{\ B\ \} \qquad (4\text{-}4)$$

$$OUT(x1,\ldots,xk,d1,\ldots,dm)\ .$$

When the procedure is recursive, we prove this assertion by using (4-4) itself as an induction hypothesis. It is only once that we must prove the partial correctness of the body.

Sequential assignment statements in (4-1) ought to be executed simultaneously as, for example, a multiple assignment statement

$$v1,\ldots,vk := w1,\ldots,wk$$

in [Gries-80]. Since w1,...,wk in the right hand sides of the assignment statements in (4-1) are fresh variables, however, the sequential execution of them is equivalent to the simultaneous execution.

As each actual variable parameter is a variable of integer type, it is one of the following variables: a simple variable (e.g. x), an integer field of a record variable (e.g. r.F), an array element (e.g. A(i)), a referenced variable of integer type (e.g. p↑), and an integer field of a referenced record (e.g. p↑.F). A weakest antecedent via each assignment statement

$$vj := wj$$

is generated by applying a transformation rule which is determined by the kind of the variable of the left hand side [Araki-79].

In this paper, since we restrict that no procedure can contain non-local variables, we have presented simple and clear verification conditions for procedure calls.

The correspondence between actual parameters and formal parameters in the verification condition (4-1) is performed by Call By Value Result, which is different from Call By Reference in the programming language Pascal. Generally the effect of Call By Value Result is not necessarily the same as that of Call By Reference. But the prohibition of non-local variables appearing in procedures makes the effects of both parameter passing mechanisms equivalent. The prohibition contributes clarity and conciseness of verification conditions.

There are some proof rules [London-78, Luckham-79, etc.] in which non-local variables can be dealt with. In those rules, non-local variables must be declared at the head of the procedures, and are treated in the same manner as parameters. In this paper, only parameters can pass the information in procedure calls. It also leads clarity and conciseness.

5. Applications

In this section we prove partial correctness assertions of procedure bodies and procedure calls. We first obtain verification conditions by applying the proof rules, and then establish the partial correctness by proving them.

Three examples are presented:

1) a non-recursive procedure which calculates the greatest common divisor of two integers,

2) a call of a factorial procedure with array elements as actual parameters, and

3) mutual recursive procedures which calculate McCarthy's 91 function.

5.1 Non-recursive procedure

A non-recursive procedure which calculates the greatest common divisor is written as the following.

```
procedure gcdproc(var g:integer;
                      x,y:integer);
var r : integer;
begin
  while y<>0 do
    begin
      r := x mod y;
      x := y;
      y := r
    end
end
```

This procedure returns the greatest common divisor g of two integers x (>0) and y (>=0).

First of all, we want to prove the following partial correctness of the procedure body.

$$x=x0 \wedge y=y0 \wedge x>0 \wedge y>=0$$
$$\{\text{procedure body}\} \qquad (5\text{-}1)$$
$$g=gcd(x0,y0)$$

We give a loop invariant condition

$$gcd(x0,y0) = gcd(x,y)$$

for the while statement in this procedure body. According to the

transformation rules in [Araki-79], three verification conditions are obtained.

$$x=x0 \wedge y=y0 \wedge x>0 \wedge y>=0 \supset gcd(x0,y0)=gcd(x,y)$$

$$gcd(x0,y0)=gcd(x,y) \wedge y<>0 \supset gcd(x0,y0)=gcd(y,x \underline{mod} y)$$

$$gcd(x0,y0)=gcd(x,y) \wedge y=0 \supset x=gcd(x0,y0)$$

Considering the properties of greatest common divisor, we can prove these three formulas true. Therefore the partial correctness of the procedure body (5-1) is established.

Now we are going to prove a partial correctness of a call of this procedure.

$$a>0 \wedge b>=0 \ \{gcdproc(z,a,b)\} \ z=gcd(a,b) \qquad (5\text{-}2)$$

The verification condition for this assertion becomes as follows.

$$a>0 \wedge b>=0 \supset (a>0 \wedge b>=0 \wedge (w=gcd(a,b) \supset [z:=w] \ z=gcd(a,b)))$$
$$\equiv a>0 \wedge b>=0 \supset (w=gcd(a,b) \supset w=gcd(a,b))$$
$$\equiv true$$

Here the variable w denotes the new value of the actual variable parameter z after execution of the call. Thus the partial correctness of this call (5-2) is proved.

5.2 Procedure call with array elements as actual parameters

Let us consider a recursive procedure below which calculates the factorial of an integer.

```
procedure fact(var x:integer; n:integer);
var t : integer;
begin
  if n=0 then x := 1
         else begin
                fact(t,n-1);
                x := n*t
              end
end
```

This procedure returns the factorial x of a non-negative integer n. We first prove the following partial correctness assertion.

$$n=n0 \wedge n\geq 0 \ \{\text{procedure body}\}\ x=n0! \qquad (5\text{-}3)$$

We construct a verification condition for this assertion, using (5-3) itself as an induction hypothesis. The verification condition is written as follows:

$$\begin{aligned}
& n=n0 \wedge n\geq 0 \\
& \quad \supset (n=0 \supset [x:=1]\ x=n0! \\
& \qquad \wedge (n\neq 0 \supset (n-1\geq 0 \\
& \qquad\qquad \wedge (w=(n-1)! \\
& \qquad\qquad\quad \supset [t:=w][x:=n*t] \\
& \qquad\qquad\qquad x=n0!)))) \\
& \equiv n=n0 \wedge n\geq 0 \\
& \quad \supset (n=0 \supset 1=n0! \\
& \qquad \wedge (n\neq 0 \supset (n-1\geq 0
\end{aligned}$$

$$\wedge\ (w=(n-1)!$$

$$\supset\ n*w=n0!))))$$

$$\equiv\ (n=n0 \wedge n>=0 \wedge n=0 \supset 1=n0!)$$

$$\wedge\ (n=n0 \wedge n>=0 \wedge n<>0$$

$$\supset\ (n-1>=0$$

$$\wedge\ w=(n-1)! \supset n*w=n0!))$$

$$\equiv\ \text{true}.$$

Therefore the partial correctness of the procedure body (5-3) is proved.

Now we want to prove the partial correctness of a procedure call with array elements used as actual parameters.

$$\begin{array}{l} A(j)>=0 \wedge i<>j \\ \{fact(A(i),A(j))\} \\ A(i)=A(j)! \end{array} \qquad (5\text{-}4)$$

The verification condition for this assertion is written as follows:

$$A(j)>=0 \wedge i<>j$$

$$\supset\ (A(j)>=0$$

$$\wedge\ (w=A(j)!$$

$$\supset\ [A(i):=w]\ A(i)=A(j)!))$$

$$\equiv\ A(j)>=0 \wedge i<>j$$

$$\supset\ (w=A(j)!$$

$$\supset\ (IF\ i=i\ THEN\ w\ ELSE\ A(i))$$

$$=(IF\ j=i\ THEN\ w\ ELSE\ A(j))!)$$

$$\equiv\ A(j)>=0 \wedge i<>j$$

$$\supset\ (w=A(j)! \supset w=A(j)!)$$

$$\equiv\ \text{true}.$$

Here the weakest antecedent of the output condition A(i)=A(j)! via the assignment statement A(i):=w is obtained by means of the transformation rules presented in [Araki-79]. Therefore the partial correctness of the procedure call (5-4) is proved.

5.3 Mutual recursive procedures

Consider the following mutual recursive procedures which calculate McCarthy's 91 function. (We owe this example to EXAMPLE 5-19 in [Manna-74].)

```
procedure p1(var z1:integer; x1:integer);
var t1 : integer;
begin
  if x1>100 then z1 := x1 - 10
            else begin
                   p2(t1,x1+11);
                   p1(z1,t1)
                 end
end;

procedure p2(var z2:integer; x2:integer);
var t2 : integer;
begin
  if x2>100 then z2 := x2 - 10
            else begin
                   p1(t2,x2+11);
                   p2(z2,t2)
                 end
end
```

First of all, we want to prove the following partial

correctness assertions of the procedure bodies.

$$\frac{\text{true}}{\{\text{procedure body of p1}\}\ (x1>100 \supset z1=x1-10) \wedge (x1<=100 \supset z1=91)} \tag{5-5}$$

$$\frac{\text{true}}{\{\text{procedure body of p2}\}\ (x2>100 \supset z2=x2-10) \wedge (x2<=100 \supset z2=91)} \tag{5-6}$$

We prove (5-5) alone, and (5-6) will be proved just the same. The verification condition for (5-5) is obtained by using (5-5) and (5-6) as induction hypotheses.

$$\begin{aligned}
&(x1>100 \supset [z1:=x1-10] \\
&\qquad (x1>100 \supset z1=x1-10) \\
&\qquad \wedge (x1<=100 \supset z1=91)) \\
&\wedge (x1<=100 \\
&\quad \supset (\text{true} \\
&\qquad \wedge\ ((x1+11>100 \supset w2=(x1+11)-10) \\
&\qquad\quad \wedge (x1+11<=100 \supset w2=91) \\
&\qquad\quad \supset [t1:=w2] \\
&\qquad\qquad (\text{true} \\
&\qquad\qquad\quad \wedge\ ((t1>100 \supset w1=t1-10) \\
&\qquad\qquad\qquad \wedge (t1<=100 \supset w1=91) \\
&\qquad\qquad\qquad \supset [z1:=w1] \\
&\qquad\qquad\qquad\quad (x1>100 \supset z1=x1-10) \\
&\qquad\qquad\qquad\quad \wedge (x1<=100 \supset z1=91)))))) \\
&\equiv (x1>100 \\
&\quad \supset (x1>100 \supset x1-10=x1-10) \\
&\qquad \wedge (x1<=100 \supset x1-10=91))
\end{aligned}$$

$$
\begin{aligned}
&\wedge(x1<=100\\
&\quad\supset((x1>89\supset w2=x1+1)\\
&\qquad\wedge(x1<=89\supset w2=91)\\
&\qquad\quad\supset((w2>100\supset w1=w2-10)\\
&\qquad\qquad\wedge(w2<=100\supset w1=91)\\
&\qquad\qquad\quad\supset((x1>100\supset w1=x1-10)\\
&\qquad\qquad\qquad\wedge(x1<=100\supset w1=91)))))\\
&\equiv((89<x1<=100\supset w2=x1+1)\\
&\quad\wedge(x1<=89\supset w2=91))\\
&\qquad\supset((w2>100\supset w1=w2-10)\\
&\qquad\quad\wedge(w2<=100\supset w1=91)\\
&\qquad\quad\supset((x1>100\supset w1=x1-10)\\
&\qquad\qquad\wedge(x1<=100\supset w1=91)))\\
&\equiv true
\end{aligned}
$$

Here w1 and w2 respectively denote the new values of actual variable parameters of calls of procedures p1 and p2. This formula is proved true by case analysis (i.e., four cases: 89<x1<=100, x1<=89, w2=101, and w2<=100). Therefore the partial correctness of the procedure body is proved.

Now we prove the following partial correctness.

$$true\ \{p1(a,100);p2(b,101)\}\ a=91\wedge b=91 \qquad (5\text{-}7)$$

The verification condition for this assertion is written as

$$
\begin{aligned}
&(100>100\supset w1=100-10)\\
&\wedge(100<=100\supset w1=91)\\
&\quad\supset[a:=w1]\\
&\qquad((101>100\supset w2=101-10)\\
&\qquad\wedge(101<=100\supset w2=91)
\end{aligned}
$$

$$\supset [b:=w2]\ (a=91 \wedge b=91))$$

$$\equiv w1=91$$

$$\supset (w2=91$$

$$\supset w1=91 \wedge w2=91)$$

$$\equiv true.$$

Therefore the partial correctness assertion (5-7) is proved.

6. Conclusion

We have presented proof rules for procedure calls, through which we obtain verification conditions. If we prove these conditions, the partial correctness of procedure calls are established. The proof rules and the verification conditions presented in this paper are so clear and concise that they are easily applicable to verification of programs with procedure calls. We can deal with mutual recursive procedures and procedure calls even with array elements, record fields, and referenced variables as actual parameters. Verification conditions can be obtained by automatic transformation of input and output conditions via statements in programs by means of the verification condition generator [Araki-81]. This is a revised version of the previous one [Araki-80] so as to deal with programs with procedure calls.

We have placed two restrictions upon the procedures. First, although procedures are able to have both variable parameters and value parameters, parameters are of integer type. This restriction comes from that of the logical system [Araki-79]. If a pointer is used as an actual parameter and the variable referenced by it appears in the procedure, some conflicts with the second restriction mentioned below will occur, because referenced

variables are to be regarded as the global variables.

The second restriction is that non-local variables cannot appear in any procedures. In [London-78, Luckham-79, etc.], non-local variables are allowed to appear in procedures. However they should be declared at the head of the procedures, and are treated exactly as parameters in the procedure call proof rules. Considering that the role of non-local variables is essentially equivalent to that of parameters, we prohibit to use non-local variables. Only parameters can pass the information on procedure calls.

These two restrictions have led clear and concise procedure call proof rules and verification conditions. They are applied to verification of programs with procedure calls more easily than those of [Apt-77, London-78, Luckham-79, etc.] are. The second restriction agrees with the recommendations to write procedures with no side effects and with information hiding. If we remove the first restriction and allow parameters of more complex data types, then a logical system for a programming language with abstract data types [Liskov-77, Nakajima-80, etc.] will be presented. To this end, we must reconstruct our logical system.

We can easily carry out proof of partial correctness of programs with procedure calls by proving the verification conditions presented in this paper. To provide a method for program verification and to establish the partial correctness of each given program are different things. Program verification can be considered as describing semantics of programs. It seems not so easy, however, to describe the meaning of each program adequately. Besides the full understanding of the programs, the progress in studies on programming languages and assertion languages

suitable for verification is required.

Acknowledgement

We would like to express thanks to Prof. T.Hayashi and Mr. K.Yoshida for their important comments on earlier drafts of this paper.

References

[Apt-77]
Apt,K.R. and de Bakker,J.W.: "Semantics and Proof Theory of Pascal Procedures," Lecture Notes in Computer Science, No.52, Springer-Verlag, 1977.

[Araki-79]
Araki,K., Hayashi,T. and Ushijima,K.: "Backward Transformation Rules for Programs with Pascal-like Pointers and Records," Trans. IECE Japan, Vol.E62, No.10, 1979.

[Araki-80]
Araki,K., Yoshida,K. and Ushijima,K.: "Implementation and Practice of a Verification Condition Generator for Programs with Pointers and Records," Trans. IECE Japan, Vol.63-D, No.10, 1980.

[Araki-81]
Araki,K., Yoshida,K. and Ushijima,K.: "Proof rules for Procedure Calls and a Verification Condition Generator Based on Them," Paper of technical Group on Automata and Languages, IECE Japan, 1981(July).

[Gries-80]
Gries,D. and Levin,G.: "Assignment and Procedure Call Proof

Rules," ACM Trans. Program. Lang. Syst., Vol.2, No.4, 1980.

[Hoare-71]

Hoare,C.A.R.: "Procedures and Parameters: an Axiomatic Approach," Lecture Notes in Mathematics, No.188, Springer-Verlag, 1971.

[Hoare-73]

Hoare,C.A.R. and Wirth,N.: "An Axiomatic Definition of the Programming Language Pascal," Acta Informatica, Vol.2, 1973.

[Liskov-77]

Liskov,B., Snyder,A., Atkinson,R. and Schaffert,C.: "Abstraction Mechanisms in CLU," Commun. ACM, Vol.20, No.8, 1977.

[London-78]

London,R.L., Guttag,J.V., Horning,J.J., Lampson,B.W., Mitchell,J.G. and Popek,G.J.: "Proof Rules for the Programming Language Euclid," Acta Informatica, Vol.10, 1978.

[Luckham-79]

Luckham,D.C. and Suzuki,N.: "Verification of Array, Record, and Pointer Operations in Pascal," ACM Trans. Program. Lang. Syst., Vol1, No.2, 1979.

[Manna-74]

Manna,Z.: Mathematical Theory of Computation, McGraw-Hill, 1974.

[Nakajima-80]

Nakajima,R., Honda,M. and Nakahara,H.: "Hierarchical Program Specification and Verification --- a Many-sorted Logical Approach," Acta Informatica, Vol.14, 1980.